Power Modeling in Database Management Systems

Data centers consume large amounts of energy. Since databases form one of the main energy sinks in a typical data center, building energy-aware database systems has become an active research topic recently. The quantification of the energy cost of database systems is an important task in designing such systems. In this paper, we report our recent efforts on this topic, with a focus on the power estimation of query plans during query optimization. We start from a series of physical models designed for evaluating power cost of individual relational operators, and use such models as a basis for a complex model for individual queries. Key parameters of the power model are generated by using linear regression tools upon running experiments with simple workloads in a static environment. To further improve the above model to make it work under system dynamics and fluctuations of workload characteristics, we develop an online model estimation scheme over the static model based on advanced modeling techniques adopted from the control engineering field. Our solution can effectively tune the model parameters at runtime to reach high accuracy in power estimation. The models are implemented in the PostgreSQL kernel and evaluated with a comprehensive set of experimental workloads generated from multiple TPC and scientific database benchmarks.

1. INTRODUCTION

Data centers, which host a lot of the world's computing resources, have been lambasted as the "SUVs of the tech world" for their enormous consumption of energy. A recent survey [Kurp 2008] shows that, in 2006, data centers in the U.S. alone consumed 61 billion kilowatt hours (kWh) of energy, which is about 1.5% of all electricity produced by the country. Meanwhile, the energy demand of data centers grows at a rate of 12% per year, leading to approximately 100 billion kWh of energy used in year 2011 - this translates into an annual electricity cost of $7.4 billion [Harizopoulos et al. 2009]. The situation in Japan is even more serious – roughly 5% of the nation's total electricity consumption is accounted for by its IT industry and this number is expected to reach 20% in 2025 [Hayamizu et al. 2011]! Naturally, energy consumption becomes one of the main optimization goals in data center design and maintenance.

Recently, energy conservation has attracted much attention from the database community, as database management systems (DBMSs) and their applications are found to be the main consumers of energy in a typical data center [Poess and Nambiar 2008]. A common theme in this research area is to design database systems with energy consumption as a first-class performance goal, as advocated by the Claremont report [Agrawal et al. 2009]. Current work in energy-aware DBMS has focused on energy-aware query optimization that considers time performance and energy usage as the target [Lang et al. 2011; Xu et al. 2010], and power management policies in distributed databases [Berl et al. 2010; Poess et al. 2010]. This paper reports our work on another key issue in the design and implementation of energy-aware DBMSs that has so far attracted less attention – modeling the energy cost of database systems.

We believe energy cost estimation in databases carries high technical significance and can be performed at two levels. At the single query level, the energy cost of alternative query execution plans is indispensable in making decisions related to query optimization [Lang et al. 2011; Xu et al. 2010] and query rescheduling [Lang and Patel 2009] in energy-aware DBMSs. An important approach to capture the power-saving opportunities in energy-aware DBMSs is to find query plans with low power cost during query optimization [Xu et al. 2010; Lang et al. 2011]. For that purpose, a query plan is explicitly evaluated by both its (estimated) energy consumption and performance. In such a scheme, accurate energy estimation is critical in ensuring the effectiveness of query optimization. At the database server or even the data center level, estimating the energy consumption of a workload is an essential part of runtime resource management toward energy conservation, especially for systems running on virtual resources [Kansal et al. 2010]. In general, power/energy estimation is known to be a critical component in any energy-aware computing system. The paradigm of adjusting hardware/software configurations at runtime to save energy is generally called dynamic power management (DPM). According to [Irani et al. 2005], all DPM policies can be divided into two groups: *predictive schemes* and *stochastic schemes*. For both cases, mechanisms to quantify system

energy consumption are needed. In this paper, we report results of our explorations in *energy cost estimation in databases, with a focus on the power estimation of query plans within the DBMS*. A salient feature of our power model is that it works under system/environmental dynamics by automatically adjusting its parameters following control-theoretic techniques. We believe that such work is required for energy-aware query optimization, and it will also build the foundation for predicting power consumption of database workloads at the system level.

Our work focuses on *power* cost instead of *energy* cost for two reasons. First, power itself has been regarded as a primary optimization goal in data center design. Due to the high density of servers in a data center, each rack is often assigned a power budget and violation of this budget renders costly business downtime. On the other hand, low power generally translates into low energy consumption. Current studies [Xu et al. 2010; Lang et al. 2011] have shown that in databases there are many query plans that require much less power with the cost of little performance degradation. Therefore, energy conservation can be achieved by harnessing such query plans. Furthermore, energy consumption of cooling systems decreases when system runs in low-power modes. A recent study [Ahmad and Vijaykumar 2010] indicates that for every watt saved from powering the servers, at least another half a watt can be saved from the cooling systems. Second, energy cost of a task can be easily obtained from its power cost and running time (i.e., energy = power $\times$ time). Note that running time estimation is a classic problem in query optimization [Chaudhuri 1998]. Existing solutions to that problem can be integrated with our work on power modeling to estimate energy consumption of database systems. By focusing on power in this work, we can effectively isolate the errors generated by our power estimation models from those inherited from time estimation by the existing query optimizer.

In this paper, we design and evaluate a two-level framework that provides accurate and robust estimations of database power consumption. We first introduce a physical model to estimate the power consumption of queries running in a database server. We build such a model from low-level models that describe the power consumption of relational operators via studying the hardware resource consumption patterns of operator processing algorithms. Parameters of such models are derived from a training query workload using classic regression tools. Such models show high accuracy in predicting query power consumption when dealing with simple workloads running under a stable computing environment. However, the value of the key parameters (e.g., number of watts to process an indexed tuple) of the model depends on system state (e.g., CPU utilization) and workload statistics (e.g., table cardinality, query arrival rate, etc.). To further improve the static model by making it adaptable to environmental and workload dynamics, we use an *online model estimation* method that uses a Recursive Least Square (RLS) estimator to periodically update the model parameters.

We implement and evaluate our power models in the PostgreSQL system. For that purpose, we develop a workload engine that produces training/testing workloads and datasets derived from various TPC and scientific database benchmarks. We validate our models using a comprehensive set of workloads generated from that engine by comparing power predictions (provided by our model) with real power measurements (using power meters). Experimental results show that our model estimates power consumption of query plans with high accuracy. As an exploratory study of this important topic of query energy quantification in DBMSs, our work focuses on power estimation for query plan evaluation and the results are tested in a single database server. Despite such limitations, we believe this work provides a general methodology that can be applied to a wide range of scenarios such as power estimation in database servers with different configurations or in systems of different scales (e.g., at server rack or entire data center level).

1.1. Problem Statement

The problem that we aim to solve is to build a powerful model that can predict the power cost of query plans in a database system. To be more specific, given a query plan, we are to quantify the *total power consumption of the server if that plan is executed*. We emphasize that such model should possess the following features:

- *Accuracy*: the model should provide accurate prediction of power consumption;
- *Robustness*: the model should maintain high accuracy regardless of fluctuations of system states and workload characteristics;
- *Fast Response*: the computational overhead of updating the model should be minimal to ensure fast response to system/workload dynamics; and
- *Non-disruptive*: the implementation of the model should not interfere with the normal operations of the DBMS.

Among the three, accuracy and robustness are obviously the key requirements, and are thus the main metrics for evaluating our models in this paper.

1.2. Contributions and Paper Organization

The database research community has just started to realize the importance of modeling power/energy consumption of query processing in DBMSs. The few reported studies on this topic have either focused on high level ideas [Lang et al. 2011] or individual aspects of database power modeling.[1] In this paper, we introduce a systematic solution with all required properties mentioned in Section 1.1. Our long-term vision is to provide useful tools and information that can help in setting up DPM policies and system performance analysis regarding energy conservation. The main technical contributions of this paper are:

- We develop a general power model for DBMS queries based on the fact that *power consumption is directly related to the amount of work*. This fact, revealed by our extensive experiments, is different from common belief.
- We develop physical models for evaluating the power costs of relational operators in a static environment. Such models can serve as the basis of more sophisticated models that handle system/workload dynamics;
- We propose an online model estimation method to automatically adjust parameters in the physical models to achieve high estimation accuracy in a dynamic environment;
- We implement our model in the kernel of a real DBMS, and evaluate it on a physical testbed using a comprehensive set of workloads generated from multiple TPC benchmarks and scientific database traces.

The remainder of this paper is organized as follows: we first compare our work with relevant reports in similar topics in Section 2; we then provide an overview of our approach in Section 3; we describe the technical details of our static power estimation models in Section 4; we present empirical evaluations of such models and discuss their limitations in Section 5; we present online model estimation method to extend the applicability of the model and evaluate it with various experiments in Section 6; finally, we conclude the paper in Section 7.

2. RELATED WORK

Green (energy-efficient) database systems. Work in energy-efficient database systems can be traced back to the early 1990's. In [Alonso and Ganguly 1992], query optimization with energy as the performance criterion is proposed within the context of mobile databases. It is unclear whether the vision proposed in that work has been implemented and evaluated in any real-world systems. In this paper, we are interested in power consumption of servers connected to the power grid, and this is a relatively new research arena in databases. Motivated by the increasing energy-related costs of database servers, the database community has only recently identified building energy-efficient database systems as an important direction of exploration [Agrawal et al. 2009]. Authors of two articles [Graefe 2008; Harizopoulos et al. 2009] share a wide range of high-level ideas on how energy efficiency can be improved in databases. Both emphasize query optimization with energy as the target – this implicitly argues for a mechanism for estimating the energy cost of a query plan. More

[1] A detailed comparison to such work can be found in Section 2.

concrete work following the path of energy-aware DBMS has also been reported. Supported by initial experimental results, [Lang and Patel 2009] presents two specific techniques to save energy in databases: controlling CPU frequency and re-scheduling user queries. In comparison to [Lang and Patel 2009], Xu and co-workers [Xu et al. 2010] design a systematic framework for energy reduction (while ensuring acceptable performance) inside a DBMS kernel. Their experiments using TPC workloads reveal the existence of many energy-efficient query plans that carry minimum performance penalty. To harness such energy-saving opportunities, they propose a cost model for evaluating query plans by both the (time) performance and energy consumption. By showing that plans of high energy efficiency coincides with those of high performance in different database environments, a subsequent report [Tsirogiannis et al. 2010] stirs up interesting discussions on whether energy-aware query optimization is a worthwhile approach towards green databases. Our opinion is that, when the search space is sufficiently big, we will find energy-efficient plans that most likely will be ignored by existing query optimizers. This standpoint is supported by more recent evidence provided by [Lang et al. 2011] and [Kunjir et al. 2012]. More details can be found in a short survey regarding energy-aware databases [Wang et al. 2011].

Other related research in green databases diverge to several directions. Poess and Nambiar [Poess and Nambiar 2008; 2010; 2011] report extensive experimental results on power consumption patterns in typical commercial database servers. Based on those results, they provide suggestions on how to make database systems energy efficient with a focus more on utilizing new hardware systems than re-implementing the DBMS kernel. Energy-related data management benchmarks are also developed: the Transaction Performance Council (TPC) officially announced TPC-Energy [Poess et al. 2010] in 2007 while another work [Rivoire et al. 2007] presents a benchmark for evaluating the energy efficiency of various sorting algorithms.

Modeling power in databases. It is worth noticing that power/energy modeling has been addressed in some of the work mentioned above. As a position paper, [Lang et al. 2011] proposes a general formula for quantifying power cost of a query plan. We explicitly use that as a starting point for our physical modeling process and perform a systematic study with extensive experiments. Furthermore, our experiments reveal the important fact that power is directly related to the number of database operations in a plan (instead of the density of such operations in time) therefore we quickly adopt a new physical model. Part of the aforementioned project in power-aware query optimization [Xu et al. 2010] is close in spirit to our work by presenting a model based on power estimation of basic database operations. Estimation error is not a main focus in [Xu et al. 2010] and therefore modeling is done in a straightforward way. Our work significantly improves the idea presented in [Xu et al. 2010] for the purpose of developing a practical framework for database power estimation and explores methodologies that are fundamentally different. First, the approach adopted in [Xu et al. 2010] is exclusively based on linear regression of experimental observations. In comparison, our method is built on the synergistic values of both physical and statistical (regression) models. Second, aiming at a robust solution that delivers high accuracy in realistic database environment, we use a dynamic modeling approach to continuously update key parameters of our model such that it adapts quickly to dynamics in the system and workload. Other technical differences/improvements also exist and we will mention those in the following sections. Another project [Rodriguez-Martinez et al. 2011] follows the same technical path as in [Xu et al. 2010] in modeling the peak power of database operations. However, the scope of their work is limited to only selection (scanning) operations while we deal with a more complete set of relational operators. More comprehensive results in modeling peak power of databases can be found in a recent work [Kunjir et al. 2012]. As peak power and average power (we are interested in) are very different concepts, the modeling processes (and apparently the models) are also different. For example, they perform an elegant analysis of the pipeline structure of query plans to identify the sources of peak power consumption and recommend plans with low peak power. Our modeling process, on the other hand, is based on a quantification of the total amount of "work" to be done in a plan. Again, only static models are introduced in [Kunjir et al. 2012] therefore it is not clear if their model handles system fluctuations or query interactions

in a composite workload. Another related work [Poess and Nambiar 2011] is concerned with estimating power consumption of industrial servers, as part of the TPC-C and TPC-Energy benchmark. It uses a pure physical modeling approach – the idea is to aggregate the peak power of hardware components in such systems.

Other related work. There are numerous reports on DPMs at the operating system level, and many DPM techniques are summarized in a survey [Benini et al. 2000]. [Karlin et al. 1990] presents randomized online algorithms for spin-block and snoopy-caching problems, which we apply to our dynamic modeling problem here in this paper. [Paleologo et al. 1998] introduces a finite-state, abstract system model for power-managed systems using Markov decision processes. Within this model, the problem of making policies becomes a stochastic optimization problem of finding optimal tradeoff between performance and power. However, the model's computational overhead is so high to an extent that we cannot endure this penalty in the gain of limited accuracy. Such work, however, inspires us toward using an online model estimation method to handle the real time disturbances that affect the accuracy of our basic power model with acceptable overhead.

Cost modeling of relational operators is a conventional problem in the database field. Work related to this topic can be traced back to the late 1970's. Initially, Astrahan and co-workers share some critical ideas in System R [Astrahan et al. 1976]. [Christodoulakis 1984] summarizes the early work and the well-accepted assumptions for query cost estimation. In [Mackert and Lohman 1986], the authors extend the work to distributed environment. Standing on their shoulders, we build up our physical models based on similar assumptions and techniques. However, since we attempt to model a different object, the variants and target objectives are no longer the same.

3. OVERVIEW OF OUR MODELING PROCESS

In a traditional DBMS [Mackert and Lohman 1986], query execution cost is treated as a linear combination of three components: CPU cost, I/O cost, and communication cost. Such costs are normally measured as the product of the number of basic operations required for executing the plan and the resource consumption in each such operation. The relevant numbers of basic operations include: number of pages read or written to disk (N_{pages}), number of tuples (N_{tuples}) to process in the CPU, and the number of bytes transmitted via networking system (N_{msg}). Such a model can be a starting point for power estimation in DBMSs [Lang et al. 2011]. Specifically, the power cost of a query plan can be expressed as

$$P = W_{cpu} \times \frac{N_{tuples}}{T} + W_{I/O} \times \frac{N_{pages}}{T} + W_{msg} \times \frac{N_{msg}}{T} \quad (1)$$

where quantity T is the query processing time, and W_X are tunable system parameters. Among them, W_{cpu} is related to the energy consumed by CPU, $W_{I/O}$ by the storage system, and W_{msg} by the networking system. In our paper, we only focus on the single server scenario and therefore the last item with communication cost can be ignored.

As a general linear model, Eq. (1) needs to be refined for accurate power estimation by considering two observations we obtain via a series of experiments we run on real database systems.[2] In such experiments, we compose a set of simple queries that include single table scan and two-table joins based on the TPC-H benchmark. At the same time, the query optimizer is forced to choose specified plans (e.g., sequential scan) for execution regardless of the costs. We measure the actual power consumption of the database server during query execution. We then feed the collected power data and proposed model into a well-known regression tool – the LP solver of the general algebraic modeling system (GAMS) software[3] – to find the best values of coefficients to adjust the model to the statistics. After a few rounds of training process, the model is established to predict the power cost to executing relational operators (More details about the model implementation are discussed in Appendix I).

[2] Here we skip the details of our experimental platform for better flow of the paper. Such details can be found in Section 5.1.
[3] http://www.gams.com/default.htm

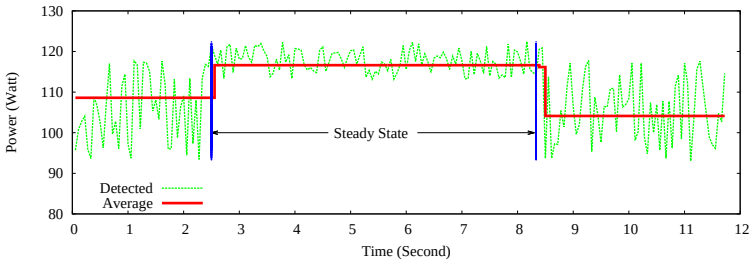

Fig. 1. Total power measurements of a database server. A workload is launched at the 2.3th second and terminates at the 8.5th second

One important note is that, in this paper, we are interested in modeling *average power* rather than *peak power* consumption studied in [Kunjir et al. 2012] and [Rodriguez-Martinez et al. 2011]. We believe average power is more directly related to energy use and thus the focus of most power-ware system research [Poess and Nambiar 2010; 2011]. A common observation from our experiments is that system power quickly enters a "steady" state in which no dramatic changes of power can be observed. However, real power consumption frequently shows fluctuations within a small range, as shown in Fig. 1. In practice, we take the average of all power readings in the lifespan of the query to smooth out the spikes and use it as the measured power in our experiments. Apparently, our model also targets at estimating such average power.

3.1. Observations on Hardware

First, there is a need to elaborate on the roles of different hardware components play in power consumption of a typical database server. To study that, we measure the power consumption of major hardware components in a database server when the system runs in idle state (when no query is being processed) and when it is fully utilized (Table I). It is easy to see that CPU contributes most to the active power used by the system (about 99%), and the difference between its peak and idle power is large (147.15-88.2 = 59 watts). All other components (e.g., hard disk, memory) consume almost the same power no matter how intensive the workload is. This is due to an important physical feature of such hardware – their leakage power dominates. Such results also verify the findings reported in other work on database energy use [Lang and Patel 2009; Tsirogiannis et al. 2010]. To further reveal the power use patterns of system components, we also record their power consumption under different workload intensities. Fig. 2 shows the results: the power consumption of CPU increases monotonically with the increase of the utilization while very little difference can be observed in the storage system. In other words, power consumption of disks is much less sensitive to the workload intensity, as compared to that of CPUs. Note that such a pattern in disks is not affected by the type of data access - for both the sequential read workload and random read workload, power consumption in the storage system is almost the same (Fig. 3).

Table I. Power consumption (in watts) of major hardware components in our database server.

Component	Peak Power	Idle Power
CPU: Xeon X5365	147.15	88.2
Memory: 4 GB Kingston	14	14
Hard drive: Seagate 2TB ST32000644NS	8.33	7.91
others	20.62	N/A
Total	190.1	111

Thus, the original model in Eq. (1) has to be modified to reflect the above findings. To estimate the power cost of a query plan, we are essentially interested in its **marginal power** if we assume the

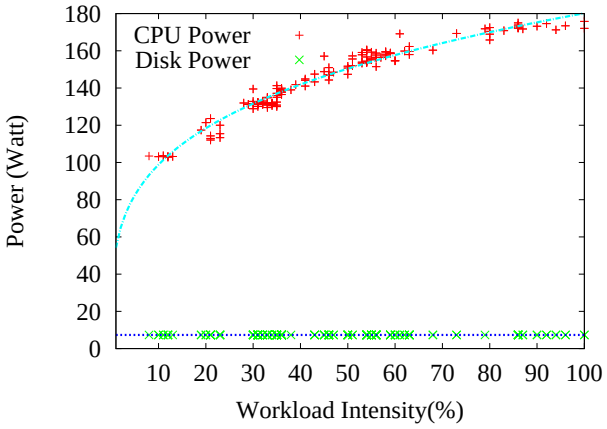

Fig. 2. CPU and hard disk power consumption under different levels of workload intensity

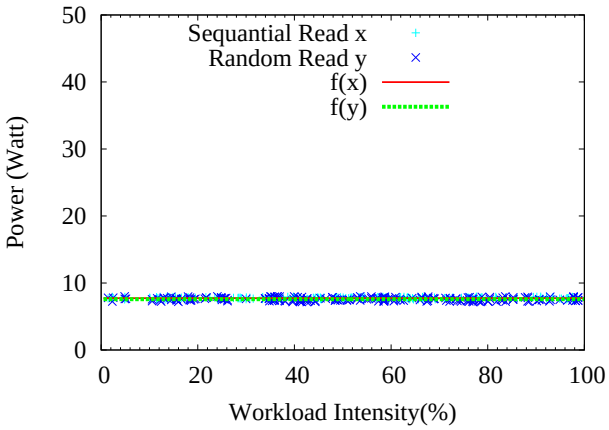

Fig. 3. Hard disk power consumption under sequential read and random read workloads

baseline power consumption is static.[4] As the CPU is the only one that contributes meaningfully to active power in a single node, we have to focus on the cost of processing tuples in CPU instead of I/O operations. In other words, we have

$$P = W_{cpu} \times \frac{N_{tuples}}{T} \tag{2}$$

Other components are ignored because their contributions to the active power are negligible. Note that the above model is significantly different from what is used in a traditional query optimizer with query processing time (or throughput) as the optimization goal [Kooi 1980; Selinger et al. 1979]. In the latter, the I/O cost is the dominating factor that often overshadows CPU cost.

[4] Here we make such an assumption explicitly and will relax it in Section 6.

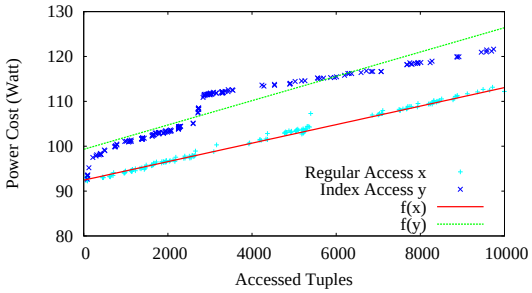

Fig. 4. Total power consumption under different numbers of tuples accessed

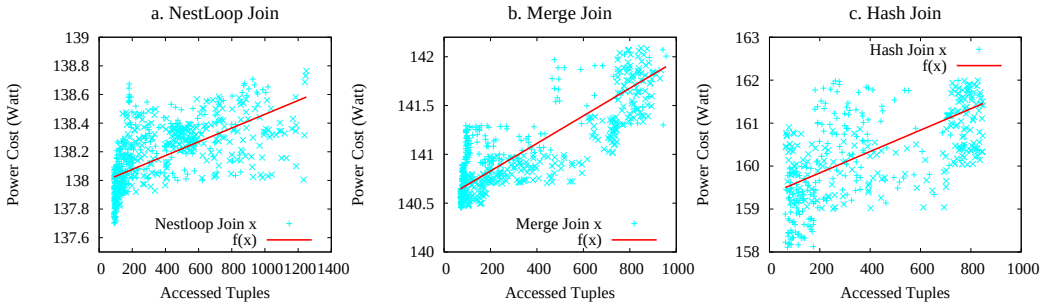

Fig. 5. Total power consumption of Join algorithms under different numbers of tuples accessed

3.2. Observations on Workload Characteristics

The model shown in Eq. (1) and Eq. (2) reflects the intuition that the total *energy* consumption of a query plan is proportional to the "work" (i.e., number of operations N_{tuples}) to be done by that plan. Therefore, power consumption is related to the work intensity (i.e., work per unit time) instead of work. For the same reason, W_{cpu} represents the energy consumption per operation. Again, the same intuition was adopted in cost-based query plan evaluation in a traditional query optimizer.

The results of an extensive set of experiments we run, however, show that the actual *power consumption of a database server depends directly on the number of database operations*. In such experiments, we run the same query multiple times under different query parameters. Specifically, by changing the range of search predicates or size of the underlying database tables, the number of tuples accessed by the query processing algorithms changes in different runs of the query. Fig. 4 shows the CPU power consumption of such runs for two queries: one with a sequential table scan and the other with indexed table scan. We can clearly see that for both queries, the CPU power increases monotonically with the total number of tuples accessed. We also run such experiments for single join queries and similar trends can be observed as shown in Fig. 5.

We believe the reasons for the above observations (which are somehow counterintuitive) are complicated. Our explanation is: reading larger number of tuples from a file needs more processing overhead such as updating the free space map and visibility map in DBMS, swapping and scheduling processes at the OS level. For example, we have observed more intensive system swap in/out activities in processing queries associated with large data than in those with small data. Such overhead translates into extra CPU power cost for reading larger number of tuples – higher cache miss rate could yield a higher power consumption as reported in [Isci and Martonosi 2003]. Therefore, we have enough confidence to modify the previous power model into:

$$P = W_{cpu} \times N_{tuples} \tag{3}$$

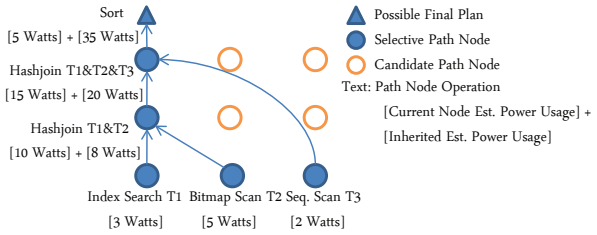

Fig. 6. One example of power cost estimation in a query tree generated by PostgreSQL

As compared to Eq. (2), the biggest difference here is that we drop the processing time T, and the physical meaning of parameters W_{cpu} is now the *marginal power cost of processing each relevant operation*. Note that the slope of the two regression lines in Fig. 4 is a good indicator of the values of such parameters, and such values will be used as initial parameters for our power models.

Given Eq. (3), the task of power estimation becomes to quantify the number of basic operations N_{tuples}, and the model parameter (i.e., unit power costs) W_{cpu}. With N_{tuples} being readily available from the existing query optimizer, the key problem is to find W_{cpu}. In the following steps of modeling, we first identify low-level factors in the workload that have an impact on W_{cpu} and treat such factors as static (time-invariant) parameters of our model via calibrating real power measurements of different workloads (Details of such work can be found in Section 4). Realistically, the model parameter should be modified under different system states and workload features. For example, Fig. 2 shows that the marginal CPU power cost levels off when the CPU utilization increases - this means our model parameter should also be much smaller when the CPU is heavily loaded.[5] We address dynamical tuning of the static model in Section 6.

4. STATIC MODELING

The theoretical model shown in Eq. (3) is too general to capture the resource consumption patterns of query plans. Specifically, since the cost of different types of operations is different (as shown in Figs. 4 and 5), keeping a single unit cost parameter W_{cpu} is obviously an oversimplified solution. Therefore, to refine the model in Eq. (3), there is a need to model the power consumption at the individual relational operator level. Note that the query optimizer in a typical DBMS (i.e., those with the System R style design) takes a bottom-up approach to build query trees, and alternative plans in each substree (representing an operator as shown in Fig. 6) will have to be evaluated. With the operator-level cost models, we can follow the same bottom-up strategy to build the cost model for the entire query plan.

Since we have identified CPU as the major active power consumer, we only need to focus on the number of tuples to be processed by the CPU. For each operator, it could be assigned with one or more power coefficients to estimate its run time power cost. In the remainder of this section, we introduce power models for a set of popular relational operators. A summary of the operator power models can be found in Table II while relevant model coefficients are listed in Table III.

4.1. Cost Models For Single Table Operations

For single table operators (i.e., *selection* and *projection*), we only consider two file organizations – heap files and index files, and their corresponding scanning algorithms – *sequential scan* and *index-based scan*. In addition, we also consider a special type of index scan – *bitmap scan* that is implemented in PostgreSQL. Since the latter involves *sorting* that is a very important component of multiple operators, we also study the cost of sorting (although it is not a relational operator *per se*).

[5] Otherwise, a plan with a huge N_{tuples} value will carry an unreasonably high power tag.

Table II. Power cost functions for relational operators.

Methods	Cost function
Sequential Scan	$w_s n$
Index Scan	$w_i n$
Sorting	$w_t n R$
Bitmap Scan	$w_i n + w_t n R$
Nested Loop Join	$w_i(n_1 + n_1 n_2 c)$
Sort Merge Join	$w_t(n_1 R_1 + n_2 R_2) + w_i(n_1 + n_2)$
Hash Join	$w_i \left(\frac{n_1}{H} + n_2\right)$

Table III. Key quantities in power estimation models.

Symbol	Definition
n	The number of tuples retrieved for CPU processing
w_s	The average CPU power cost for processing a regular tuple
w_i	The average CPU power cost for processing an index tuple
w_t	The average CPU power cost for sorting a tuple
H	The number of hash partitions
R	The number of runs in a sorting algorithm
c	Selectivity of join condition

Sequential Scan. Sequential scan is a scan method that each row of source table is read in a sequential order and relevant columns are checked against a predicate. The anticipated power cost of scanning a table with n tuples, according to Fig. 4, is $w_s n$.

Index Scan. Index scan is similar to sequential scan except an (tree-based or hash) index is used to reduce the number of tuples accessed. Thus, the estimated power cost for index scan is $w_i n$ for searching in the n retrieved tuples. Note that the unit power cost of accessing an indexed tuple w_i is different from that of a tuple in sequential scan (w_s).

Sorting. Sorting is an operation that is "hungry" for CPU resource due to the need to process the whole table in multiple runs. In estimating the sorting power cost, table size and the specific sorting algorithm are the key factors to consider. For the merge sort algorithm implemented in PostgreSQL (other database systems may be different), the power cost for sorting is $w_t n R$, where n is the number of tuples fetched to be sorted and R is the number of runs the sorting algorithm encounters. Note that R is a logarithmic function of n with a large base such that it is generally a small number even for large database tables.

Bitmap Scan. Bitmap scan has significant space and performance advantage over other index structures for data that contains very few distinct values in the column of interest. Bitmap index answers queries by performing bitwise logical operations based on bit arrays (commonly called bitmaps). The scan is based on bitmap index on the records. Since bitmap scan also needs sorting of the intermediate results, the cost is $w_i n + w_t n R$.

4.2. Cost Model For Join Methods

For any two table joins (original or temporary table), the power consumption depends on the join algorithm used.

Nested Loop Join. The cost model is $w_i(n_1 + n_1 n_2 c)$ where c is the selectivity of the join condition, n_1 and n_2 stand for number of tuples fetched from the outer table and the inner table, respectively. The nested loop join will pick up a tuple from the outer table and try to find matching tuple(s) from the inner table. For each tuple in the outer table, $n_2 c$ tuples from the innder table will be accessed (n_2 tuples accessed in worst case). Thus, the total number of tuples accessed would be the sum of number of outer table tuples accessed and their associated inner table tuples. Therefore,

the CPU power cost is $w_i n_1$ for the outer table[6] and $w_i n_1 n_2 c$ for accessing the inner table. Note the existing query optimizer will provide us with the estimated value of c.

Sort-Merge Join. The power cost of merge join comes from the two stages of the algorithm: *sorting* and *merging*. First, the sorting cost for both tables can be calculated using the aforementioned sorting cost model. For merging, the power cost is the cost of scanning each tuple in both sorted tables. The total number of tuples accessed at this stage is $n_1 + n_2$. Therefore, the cost model for merge join is $w_t(n_1 R_1 + n_2 R_2) + w_i(n_1 + n_2)$ where R_1 and R_2 are the numbers of runs in sorting the two tables.

Hash Join. The hash join algorithm takes hashing to both relations on the join attribute and identifies the outer and inner tables in a subsequent probing phase. The power cost of hashing is the sum of power costs from outer table access within each hash partition and inner table access. Thus, the metric is $w_i \left(\frac{n_1}{H} + n_2\right)$.

4.3. Putting Everything Together

The above models can be combined to form complex models for any arbitrary query plan. For example, for the query execution path tree shown in Fig. 6, its total power cost can be generated from adding the power cost from each node (the relational operator) in the tree along the path. As a result, the total power cost is the sum of those from two hash joins, three scans and one sorting operation. To be specific, following the path shown in Fig. 6, we can generate the following model to quantify the power cost of the plan:

$$\begin{aligned} P &= w_i n_1 + w_i n_2 + w_i n_2 R_2 + w_s n_3 + w_i \left(\frac{n_1'}{H_1} + n_2'\right) + w_i \left(\frac{n_{12}'}{H_{12}} + n_3'\right) + w_i n_{123}' R_{123}' \\ &= w_s n_3 + w_i \left(n_1 + n_2 + n_2 R_2 + \frac{n_1'}{H_1} + n_2' + \frac{n_{12}'}{H_{12}} + n_3' + n_{123}' R_{123}'\right) \end{aligned} \tag{4}$$

where n_X stands for the number of tuples fetched from table X, n_X' stands for the number of tuples generated from previous relational operator, and H_X is the number of hash partitions in the relevant hash joins. Note that values of a few quantities, including n_X, H, and R, are provided by the existing query optimizer.

5. STATIC MODEL VALIDATION

5.1. Experimental Setup

In this section, we mainly introduce our experimental setup, including hardware and software specifications, and how we evaluate our model's effectiveness in power cost estimation.

Hardware. Our testbed for model validation consists of two servers for different purposes. The one called "worker" contains a 3.0 GHz quad-core CPU Xeon X5365, 4GB of 667 MHz DDR3 memory, and a 2TB 7200RPM hard drive, as shown in Table I. It is used to run the DBMS and thus the target for power modeling and estimation. The other one called "monitor" is responsible for runtime collection of experimental data including query statistics and power consumption. Power measurement is done by power meters (i.e., USB Oscilloscope DSO-8502 and watts' Up power meter) attached to "worker" and linked to the "monitor" via USB connections for data reading. Specifically, the DSO-8502 is used to measure CPU power and the watts' Up for that of the entire database server.

[6] Actually, this part of the model can also be $w_s n_1$ if the outer table is accessed by a sequential scan. Here we just assume the outer table is accessed via indexed scan. The same assumption applies to inner table access and other join algorithms.

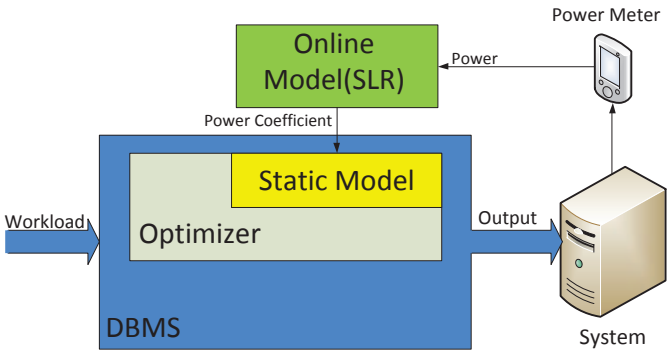

Fig. 7. A sketch of the experimental platform

Software. Our "worker" machine is installed with PostgreSQL 8.3.14 as the DBMS under Ubuntu 9.10. The DBMS's kernel was hacked to provide detailed runtime system information such as estimated cost, data histogram and plan selection. Along with the DBMS, we also implement (and deploy in the "worker") a workload generator based on TPC-H, TPC-C benchmarks[7] and datasets from the SDSS database[8] – a well-known large-scale scientific database. The workload generator creates a comprehensive set of workloads designed to simulate the effects of major factors that impact power modeling, and are used to test the effectiveness of our models. The software architecture of the "worker" server is sketched in Fig. 7.

Experimental Setup. We conduct a series of experiments to verify the models mentioned in Section 4. As the first step of the modeling process, we create an ideal environment to run experiments such that we can concentrate on verifying the model structures and obtaining initial values for the model parameters from a set of training workloads. In this setup, we follow the assumptions mentioned in Appendix III to create such an environment in which the numbers of operations given by the query optimizer are accurate.

We use the dedicated "worker" server for our experiments and feed the database with simple workloads that consist of very few types of queries. In such experiments, we set the multi-processing level (MPL) to one, i.e., only one query is processed at a time in the DBMS. We measure the real power of the entire server and compare it with the estimated power given by the corresponding models. A metric named *Estimation Error Rate* (EER) is used to quantify the model accuracy. Specifically, EER is defined as

$$EER = \frac{|C' - C|}{C'} \tag{5}$$

where C stands for the estimated power given by our model and C' is the actual steady-state power consumption (see Fig. 1) of the whole server measured by a power meter.

5.2. Experimental Results

5.2.1. Results of Single Table Models. We generate a set of 20 data files with a total size of 2.1TB (w/wo indexes) for single table scans. Moreover, we prepare a set of similar queries (queries with the same structure but different selection predicates) to test the table scan models. For each scan operation, there are equal number of queries visiting large table files (e.g., 6GB *lineitem* table in TPC-H) or small table files (e.g., 10MB *order* table in TPC-H). Each experiment is repeated 100 times and we compute the average EER in all 100 runs. The results are plotted in Fig. 8.

[7] http://www.tpc.org/
[8] http://www.sdss.org/dr7/

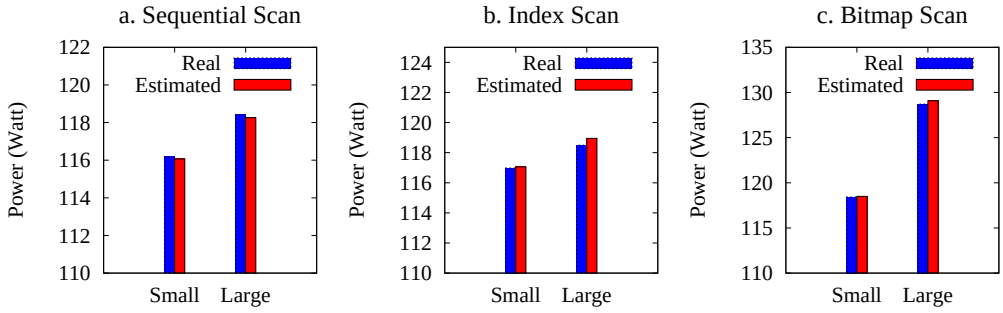

Fig. 8. Empirical validation results of single table operations' power models.

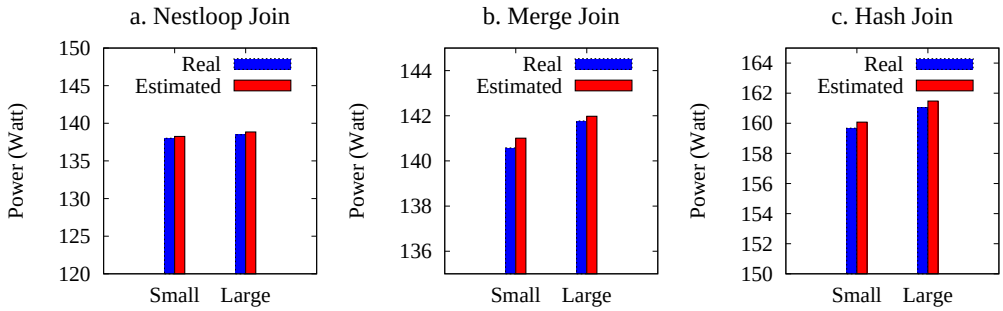

Fig. 9. Empirical validation results of multi-table operations' power models.

As seen from Fig. 8, the estimated power shows very little difference to the real power measurements in the sequential scan experiments (i.e., $\pm$ 0.3 watts). This translates into an average EER of less than 0.5% for both large and small file scans. We believe this is a high accuracy as some random error is inevitable – errors lower than 0.5% are essentially negligible. Results for index scans (bitmap and regular index scan) are also very promising. The estimated power cost is nearly the same as real power measurement in all cases ($\pm$ 0.5 watts). In summary, our static model works very well for single table operations by achieving accuracy that is over 99%. One other thing to point out is that the power consumption is always higher in cases of large file access than in small files, thus supporting our conclusion that power consumption depends on the amount of work to be done (Section 3.2).

5.2.2. Results of Multi-table (Join) Models. The verification of join power models uses the same experimental setup and data files in previous experiments (Section 5.2.1). From the TPC-H official tool, we create a set of queries with one single join of two tables. Each join operation is a combination of two table scans and one join operation (see Fig. 6). It is not a surprise that system power increases much faster than scan operators when the size of the joined tables (thus resource consumption) increases (Fig. 9). Again, the observed EERs are less than 0.5%, indicating the effectiveness of the join models.

5.2.3. SDSS Validation Experiments. To test our models under very large datasets, we materialize a database from the published SDSS data.[9] Then we create three workloads: workload I is an equality search based on a sequential single table scan; workload II is a merge join of two tables after

[9] Release 7, URL: http://www.sdss.org/dr7

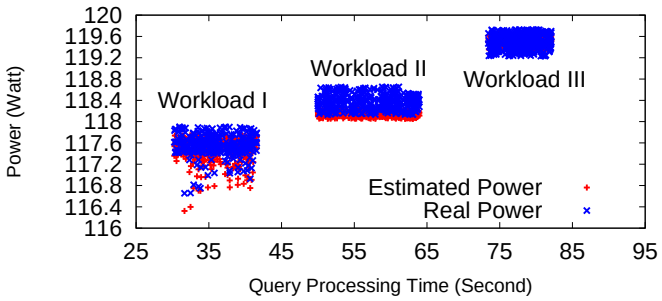

Fig. 10. Power consumption and processing time of three SDSS workloads

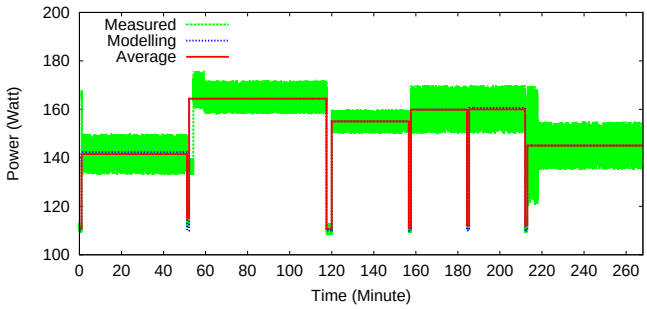

Fig. 11. Power model validation using a batch TPC-H workload

range searches; and workload III is a range search based on sequential scan. We run each single-query workload for 1,000 times with different search predicates generated randomly, and the results are shown in Fig. 10. The size of the largest table scanned in such queries is 2TB. As we can see, the difference between the estimated power and measured power is very small in almost all cases. The average EERs of the three experiments are 3.32%, 2.66%, and 1.78%, respectively.

5.2.4. Composite Model Validation. In the final set of experiments, we use composite workloads that contain a random subset of the TPC-H queries. As compared to those mentioned in Section 5.2.2, such queries include joins of more than two tables combined with table scans. Therefore, the composite model as that shown in Section 4.3 will be constructed to estimate the power cost. Fig. 11 shows the power measurements and estimation for a period of time during executing the random TPC-H queries. We can hardly see any difference between the estimated power (dashed blue line) and the average of measured power (solid red line). In fact, the EER in this experiment ranges from 0.32% to 2.65%. We also run single-query workloads for all 22 queries of the TPC-H benchmark, and the estimation results are demonstrated in Fig. 12. Again, we observe very high accuracy, with an average EER of 2.97% and the highest EER reaching only 4.38%.

5.2.5. Limitations of the Models. The above experiments show that the power prediction using our models is very successful in a static environment. However, our modeling task is far from complete. When we change workload features and system resource availability, our models can easily fail. We first test the system with 9 different composite workloads generated from TPC-H under an MPL that is greater than one. Specifically, we initialize 100 client threads and allow queries sent from all clients run concurrently (i.e., MPL can reach 100). The results of such experiments are plotted in Fig. 13 – we can see that the EER for all workloads are over 40%, with the highest one reaching 65%. These are very inaccurate estimations as they almost reach the upper bound of possible errors,

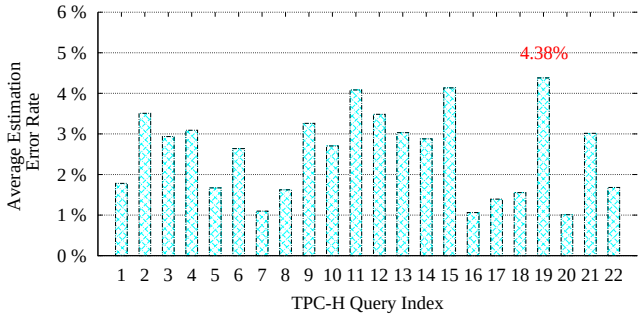

Fig. 12. Power estimation errors upon running 22 queries in TPC-H

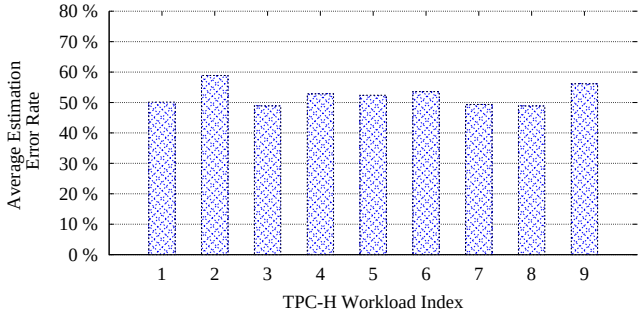

Fig. 13. Power estimation error for a mixed TPC-H workload with nine different query sets under an MPL value of 100

such bounds are described in Appendix II. Note that all absolute errors in Fig. 13 are negative, meaning the model systematically underestimates the power cost of the workloads.

In another experiment, we emulate a change of CPU resource availability by introducing CPU-intensive non-database jobs into the system. Specifically, we fork a process that creates a number of child threads to compute the Fibonacci sequence. Again, this causes a serious underestimation of power consumption (blue line in Fig. 14) and an average EER of 65%. This clearly shows that our model needs to be updated dynamically to capture the changes of system status. One might argue that the problem is caused only by the models' failure in capturing the increased baseline power of the system, and can be easily solved by reading in a baseline power in real-time. However, such a solution would still not be robust. First, competition between concurrent queries has profound effects on power consumption. Second, when such effects are mixed with those caused by system states (such as that in Fig. 14), it is almost impossible to tell them apart. In the context of this experiment, we implement and test such an *ad hoc* solution that measures the system power consumption in real-time and add the measured power to the results of the static model as the estimated total power. The results (pink line in Fig. 14) clearly show that the *ad hoc* model systematically overestimates the power with a large error margin (i.e., average EER reaches 18.67%).

5.2.6. Source of Errors. Many factors can contribute to the errors in power estimation of database queries. According to our empirical study mentioned above and existing wisdom from the software engineering community,[10] such factors can be put into three categories.

- *System status*. Run time state change of the database system and even the OS can cause significant errors in power estimation. A simple example can be seen in Fig. 2: the marginal increase of CPU power is not linear to the CPU utilization. In other words, the parameters w_s, w_i and w_t should

[10] A summary can be found at http://eu.wiley.com/legacy/wileychi/hbmsd/pdfs/mm154.pdf.

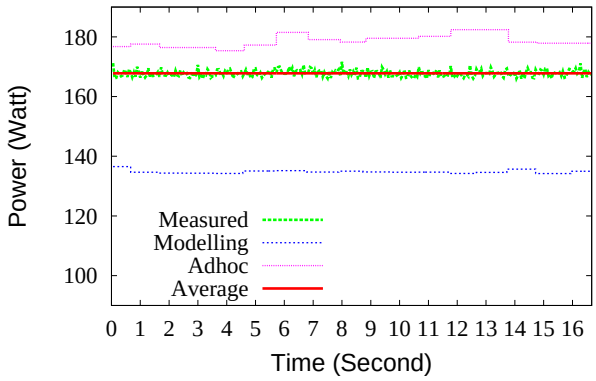

Fig. 14. The static model fails under a change of system contention

have different values under different system contention, and the change of the latter cannot be easily detected. Other events such as initialization of competing computational jobs can cause the same problem;

- *Operation quantity estimation error*. Errors are inevitable in estimating the number of input tuples (e.g., n_1, n_2 in Table II) for a relational operator in the plan. In our models, such values are provided by the existing query optimizer and are inherently inaccurate. In fact, this is a classical problem in query optimization as the query processing time also depends on such quantities [Chaudhuri 1998].
- *Workload dynamics*. A workload generally contains queries with different resource consumption patterns and the interactions among concurrent queries are very complex. Workload features may change over time and significantly impact power cost by changing variables such as cache hit rate and concurrency level.

To derive an accurate model that minimizes the above errors, we could integrate all relevant factors into an augmented physical model. However, this is an infeasible solution because it is impossible to locate all the possible factors, and model their effects on power consumption. The solution we propose and implement here is to use an online feedback mechanism that adjusts the weight parameters of each relational operator's estimation function by capturing the recent trends of system and workload dynamics in order to decease the EER.

6. DYNAMIC MODELING

In this paper, we propose an *online model estimation* strategy to minimize the errors. The main idea is: we keep the structure of the previous physical model we develop. We treat the database system as a black-box and model the cost parameters (w_s, w_i, and w_t) in our existing model as system-level variables whose values reflect the combined effects of all possible system/environmental factors. We then use a feedback control based mechanism to periodically update the parameters using real-time power measurements. As a result, errors generated by the sources mentioned in Section 5.2.6, even those in operation quantity estimation, will be compensated for by adjusting such parameters.

For each relational operator, we define its unit power consumption as a function of time (instead of a time-invariant model shown in Table II) since we need to adjust the model in response to the dynamics in system/environment. Such models are updated in every period with length T_s to refine the estimation results. In each sampling period, we collect the real power measurement and use it to compute the estimation error and average baseline. Whenever a query enters the system, the online model will use those two pieces of information to update the power coefficients which will, in turn, be used for power cost prediction of the query plans.

6.1. Online Model Design

In this section, let us introduce the online model estimation scheme, which is based on the a refined Recursive Least Square (RLS) estimator [Wang et al. 2008; Vahidi et al. 2005; Wang et al. 2009] with directional forgetting according to our preference.

We will maintain an operation vector $\vec{n} = \{n_1, n_2, \cdots, n_m\}$ to hold quantities of all operations of queries currently being processed in the system. Recall that the values in $\vec{n}$ are provided by the query optimizer. In general, the RLS scheme requires a vector $\vec{w} = \{w_1, w_2, \cdots, w_n\}$ to hold all the parameters to be updated and a variable $k = \sum_{j=1}^{n} w_j$. In our case, we have $\vec{w} = \{w_s, w_i, w_t\}$ and $k = w_s + w_i + w_t$. We then define a new vector $\vec{w'} = \{w_s, w_i, w_t, k\}$, and denote the value of $\vec{w'}$ at period j as $\vec{W}(j)$. Similarly, for the operation vector $\vec{n} = \{n_1, n_2, \cdots, n_m\}$, we define another vector $\vec{n'} = \{n_1, n_2, \cdots, n_m, 1\}$ and denote the value of $\vec{n'}$ at period j as $\vec{N}(j)$. At each period, the actual power consumption of the server, $\mathcal{P}$, is measured. The RLS model generates a quantity $p(j)$ as the baseline power from the measurements of the last $j-1$ periods and current $\mathcal{P}$ as follows:

$$p(j) = \frac{((j-1)p(j-1) + \mathcal{P} + P_I)}{j} \tag{6}$$

where P_I is idle power for all hardware components except the CPU, which is $111 - 88.2 = 22.8W$ in the server we used according to Table III. We also set the initial value of the baseline power to 111 watts. In other words, we have $p(0) = 111W$. Now we know how to refine the baseline power to estimate total power cost. The next step is to use this data to adjust the model to accurately estimate power cost of query plans. The parameter vector $W(j)$ is updated as follows:

$$\vec{W}(j) = \vec{W}(j-1) + \frac{e(j)\vec{N}^T(j)\mathbf{M}(j-1)}{\lambda + \vec{N}(j)\mathbf{M}(j-1)\vec{N}^T(j)} \tag{7}$$

where $e(j) = p(j) - \vec{N}^T(j)W'(j)$ is the estimation error, $\mathbf{M}(j-1)$ is the covariance matrix of vector $\vec{N}(j)$, and λ is the constant forgetting factor within $[0, 1]$ – a smaller λ enables the estimator to forget the history faster. The following routines are invoked at the beginning of every period j of model updating:

(1) the RLS estimator records the operator vector, $\vec{N}(j)$ and calculates baseline power $p(j)$;
(2) it computes $\vec{W}(j)$ according to Eq. (7).

The RLS estimator adapts itself so that $e(j)$ is minimized in the mean-square sense. When the two variables, $\vec{n}$ and $p(j)$, are jointly stationary, this algorithm converges to a set of tap-weights which, on average, are equal to the Wiener-Hopf solution [Lawrie 2007]. As a recursive algorithm, the RLS estimator has very low computational overhead (tens of microseconds as we recorded in our experiments). It is also robust against different workloads and system conditions. In our method, the initial values for the power parameters (w_s, w_i, and w_t) are based on results obtained from the static models upon running a composite workload (see Appendix I for details). We conduct power statistics identification experiments for all the TPC-H queries, and the initial value of the parameter vector is $\vec{W}(0) = \{0.0023178, 0.0024535, 0.00223659\}$.

One special note about $p(j)$ is: instead of being measured via a power meter, quantity $\mathcal{P}$ can be translated from CPU utilization (provided by the OS) following the regression curve shown in Fig. 2. The advantages of this method are obvious: first, real measurements from a power meter are always associated with delays from the communication channels and resource overhead; Second, our database system can be deployed in a server without a power meter connected - they are only required for generating the initial values for the model parameters. This is a huge benefit in data centers that host a large number of servers. By comparing this method with real power measurements we found that the differences are negligible.

The length of T_s implicitly affects the accuracy of the dynamic model. It is relevant to the frequency of incoming query request. If the query arrival rate is high, T_s will be set a smaller value to sampling sufficient variance. Otherwise, we could make it longer to avoid possible disturbance and computational overhead. In our experiment, we set it to be 1/18 second, such a sampling frequency is the same as that of the system power measurement.

6.2. Online Model Validation

6.2.1. Experimental Setup. The experiments are run in the same environment as that mentioned in Section 5.1. However, as the enhanced model is meant to solve more complex problems, our workload generator produces datasets and workloads that create scenarios that our static models cannot handle well. First, the workload generator borrows data and queries from three sets of benchmarks:

(1) The generator produces a query pool that consists of 2,000 queries derived from the 22 standard queries in the TPC-H benchmark by changing the query parameters. The workload generator draws queries from such pool with a predefined distribution of query arrival time and features such as the level of resource sharing, query priority, and multiprogramming level;
(2) We also use a 1TB SDSS database that includes 53 million unique astronomical objects such as stars, galaxies, and quasars. The set of 400 queries against this database are extracted from the query templates posted on the SDSS website – it mainly consists of large table scans and joins of few tables (mostly two-table joins). The purpose of using this workload is to identify the model's capability of estimating power cost of a large database.
(3) Finally, we use a TPC-C benchmark tool named TPCC-UVa[11] to generate OLTP workloads. Note that TPCC-UVa forms a black-box testing environment for our model verification as it is a closed benchmark tool in that users cannot access (let alone modify) the queries.

Based on the queries from the above three benchmarks, we design a series of experiments running different database workloads. In all such experiments, we set the MPL to 100 to create a realistic database runtime environment in which multiple queries are processed concurrently. Each type of workload is for verifying our dynamic model's ability to handle one (or more) category of errors mentioned in Section 5.2.6. By changing workload parameters, we can simulate different levels of impacts the error sources have on our power models. Particularly, we have the following three types of workload.

(1) *Type I*: To test the accuracy of our model under workload error and system error, we define this type of workload with different levels of resource sharing among concurrent queries. Specifically, in such workloads, we simulate the *share-everything* and *share-nothing* patterns in what we call the *fine-grained parallel* and *coarse-grained parallel* workloads. The coarse-grained parallel workload contains queries of large computational overhead, and little data shared with other queries. On the other hand, fine-grained parallel workload is generated by using queries of small computation, large interaction and considerable amount of data shared among queries. In a share-nothing system environment, the power cost of the system varies rapidly over time due to the large volume of page demand and replacement. In the share-everything environment, the power consumption may be more stable since most data are reusable in the cache and the CPU time won't be wasted for waiting I/O. This type of workload is generated from the TPC-H query pool as mentioned above.
(2) *Type II*: The resource estimation error is another important factor that causes the power estimation failure. Poor estimation of data distribution (in the form of data histograms) in the database tables is the main reason for that [Chaudhuri 1998]. In order to verify its effects on the accuracy of our model, this type of workload contains either: (i) *deterministic access* (DA) queries that visit similar regions in the data domain from time to time; or (ii) *random access* (RA) queries that randomly touch all spectra of the data domain. In running the DA workload,

[11] http://www.infor.uva.es/~diego/tpcc-uva.html

the query optimizer will quickly learn the data distribution after running very few queries and resource estimation for following queries will be accurate. For the same reason, in running the RA queries, the data histogram will be updated frequently and it leads to inaccurate resource estimation. In short, the purpose of this workload type is to test our model in facing the resource estimation errors, or more specifically, errors in selectivity estimation. This type of workload is also generated from the TPC-H query pool.

(3) *Type III*: In a real-world database server, processes other than the DBMS will run concurrently and such processes may cause fluctuations of the system's resource availability. We design this type of workload to emulate those changes by via two mechanisms. The first one is to introduce a user program that automatically spawns new processes to compute Fibonacci sequences. It is a pure CPU intensive program and the number of its child processes can be fixed or limited to avoid thread bomb and produce different resource consumption patterns. The other mechanism is to change CPU frequency at runtime via the Dynamic Frequency and Voltage Scaling (DVFS) technique built into many modern CPUs. Both mechanisms can introduce significant changes of system capacity. Thus, this workload type is for verifying our models under system error. Queries in this type of workload are drawn from both the TPC-H and SDSS query pools.

We compare the performance of the dynamic models with the static models we develop in Section 4.3 and the *ad hoc* model mentioned in Section 5.2.5. Note here, since we have multiple queries running in the system now, the average EER that we used to evaluate the system's performance is redefined as the arithmetic mean of the EERs of all involved queries.

6.2.2. Experimental Results.

Results of Type I workload. In such experiments, we create a query pool for the type I workload sets following the descriptions in Section 6.2 and randomly pick queries from it. Specifically, we try 9 workloads, each draws a different number of queries randomly from the TPC-H query pool and the query set size ranges from 10 to 2,000. As shown in Fig. 15, the EERs generated by the RLS models are significantly smaller than those under the static models for both Type I workloads (i.e., fine-grained and coarse-grained parallel). Also, by comparing the results of two workload types under the RLS model, we can see that our model handles the coarse-grained parallel (i.e., share-nothing) workload as well as the fine-grained (i.e., share-everything) parallel workload, with an all-round average EER of 8.89% and 6.93%, respectively. The highest EER recorded for these two workload types are not much higher – they are 10.88% and 9.07%. The fact that both query patterns leads to similar model accuracy shows that our model can effectively handle the interactions among queries. The *ad hoc* solution has much better accuracy than the static models, showing its ability to partially capture the interaction among queries and reflect it in the modeling. However, it is still no comparison to the full-fledged dynamic model – its EER often doubles or even triples that of the latter (average EERs are 28.54% and 34.78%, respectively). Another observation here is, in majority of the experiments, the coarse-grained parallel workload causes larger errors than the fine-grained parallel workload. This shows that errors caused by query interaction is a major obstacle to overcome in database power modeling.

Results of Type II workload. As shown in Fig. 16, the dynamic model shows a very high accuracy while handling DA workload – the average EER is 8.93% with the highest EER being 11.9%. For the DA workload, queries always visit the same part of the table therefore it leads to very high cache hit rate. In other words, the workload runs under a relatively steady system environment. That is likely to be the reason why the static model shows similar estimation errors in almost all experiments (i.e., accuracy stays around 40%). When it comes to RA workload, although the query optimizer could produce large errors in resource estimation, our dynamic model can capture the trends of such errors and compensate for them. The EER is again lower than 10% for most of the cases – the average EER is 8.26% with the highest EER being 10.07%. Similar to the Type I workload experiments, the *ad hoc* model is superior to the static model in modeling accuracy but demonstrates much higher EERs than the dynamic model.

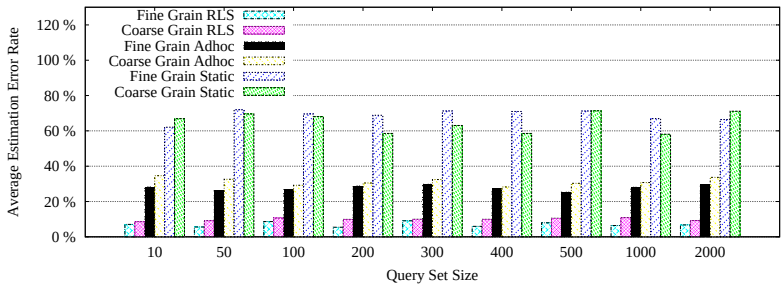

Fig. 15. Model accuracy under different Type I workloads with different data sharing patterns

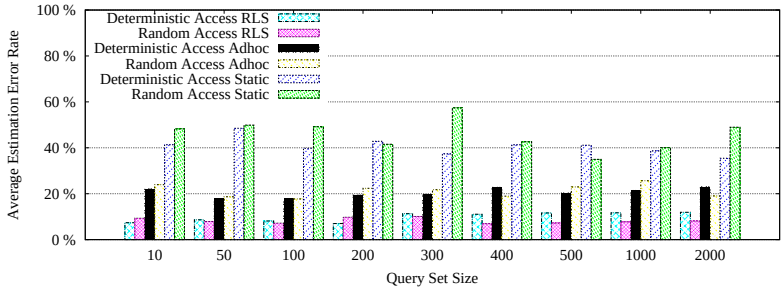

Fig. 16. Model accuracy (average EER) under different Type II workloads with different data access patterns

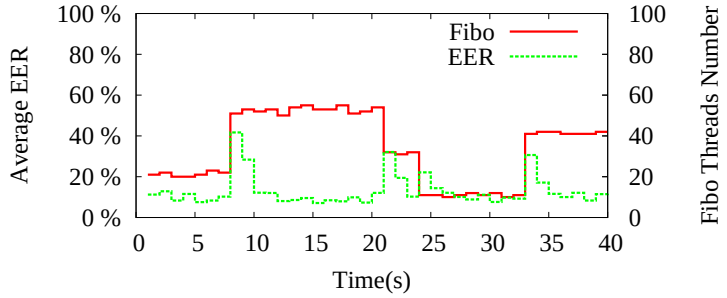

Fig. 17. Model behaviour at runtime when other Fibo processes "steal" system resources

Results of Type III workload. As discussed above, we create a pure CPU-intensive process called Fibo, which continuously calculates numbers in the Fibonacci sequence. We use Fibo to emulate real-world scenarios in which non-DBMS processes compete with the DBMS for resources. At runtime, Fibo can automatically spawn different numbers of child processes according to parameters received from a communication channel. The results of this experiment are demonstrated in Fig. 17: the system starts with 20 Fibo processes and this number is changed to 54 at the 7th second and drops at the 18th and 24th seconds to 10, and then increases to 40 at the 33rd second. By Comparing the changes of number of Fibo processes running in the system (red line) and power estimation errors, we can see that our online model can capture the trends of such changes and react within a short period of time (i.e., shorter than 3 seconds in this experiment).

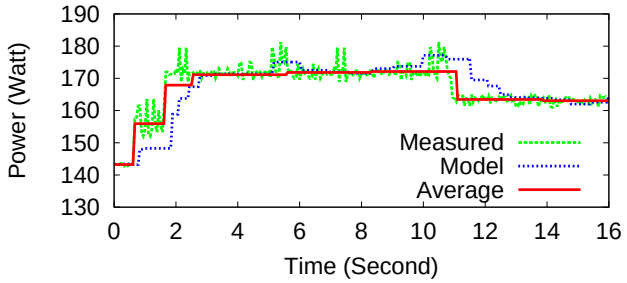

Fig. 18. Model behaviour at runtime upon sudden changes of DVFS level

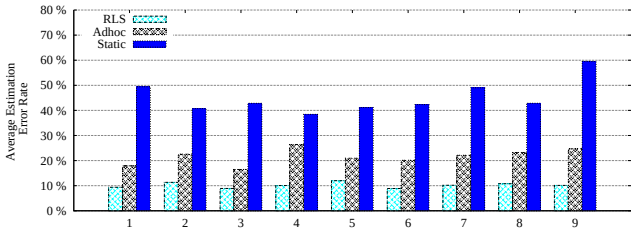

Fig. 19. The average EER of running an unknown TPC-C workload repeatedly

We also emulate sudden changes of the system states by changing the CPU power consumption pattern at runtime. We take advantage of the DVFS feature of the CPU to implement that. In Fig 18, we present a sequence of power data collected on-the-fly by plotting the measured power as the green line, and the estimated power as the blue line. At first the CPU runs at 70% frequency, with a power consumption of the system around 143 watts when running a mixed TPC-H workload. In about one second, the CPU frequency increases to 80%, and then jumps to 90% after another second. Our model starts to adjust the parameters to catch the changes of CPU frequency immediately after the first jump of frequency, and manages to minimize the estimation error at about one second after the second jump. The system experiences a sudden decrease of power at the 11th second, and the RLS model, again, is able to respond to that in about two seconds' time. The average EER we record during this whole period is about 9.72%. This complements the above experiment in showing our model's ability to deal with dynamics in system resource availability.

Black box validation. We also validate our model within a closed query environment generated from a non-commercial TPC-C tool called TPCC UVa, as mentioned before. Since we cannot change data distribution or the query composition inside the workload, this serves as a perfect tool for black box testing. As seen in Fig. 19, the average EER of our dynamic model is around 10% in a 10GB DBMS configuration compared to the static model's EER of 51.4% and the ad hoc model's 23.62%. This clearly shows that our RLS model is robust even under a workload whose internal features are concealed.

Effects of forget factor λ. Another interesting experiment is to see the effects of the important forget factor λ in our dynamic model. The results of this experiment could help us fully explore the effectiveness of the model and make recommendations on such value of λ. For that purpose, we create six workloads – two from SDSS queries and four with various combinations of TPC-H queries – to provide a diversified testing environment. As seen in Fig. 20, the forgetting factor λ could affect the accuracy of the RLS model significantly. For the SDSS workloads, the results are stable without showing much difference under different λ. Our explanation is: most queries in SDSS are I/O-bound due to the sheer size of the database table - this creates a very static situation

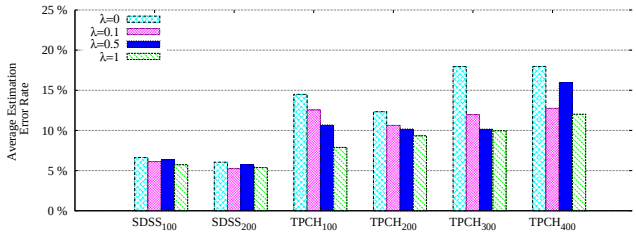

Fig. 20. Accuracy of the RLS model under 6 different workload sets and 4 different λ values

in terms of power consumption. However, in a dynamic environment, if the system states follow a historical trend (e.g., those of coarse-grained parallel workload or a deterministic access workload), increasing the value of λ gains significant benefits in terms of model accuracy. Thus, we suggest the maximum value (i.e., 1.0) of λ be used in order to obtain more historical data and higher accuracy of the model.

6.2.3. Discussions. We are in general satisfied with the performance of our models. First, the static model indeed provide estimations of the cost of query operators with negligible errors. For more realistic database workload and environment, the RLS-based model is effective – EER is in the range of 8-13%. Note that cost estimation in composite workloads is such a difficult problem that an estimation is regarded satisfactory when it is of the same magnitude as the real value [Waas and Galindo-Legaria 2000]. Power estimation, on the other hand, is not as difficult due to the narrow bound of errors (see Appendix II). Nevertheless, the estimation error can easily approach the boundary value(s) if the model is poorly designed (Section 5.2.5). The absolute errors of our dynamic model are less than 8 watts in a server with a saturation power of 190 watts and active power range of 80 watts. A comparison between our work and others is less meaningful than one might think: [Kunjir et al. 2012] and [Rodriguez-Martinez et al. 2011] focus on peak power and do not provide dynamic models while [Xu et al. 2010] does not report model accuracy at all. By just reading the numbers, our models have lower errors than those reported in [Kunjir et al. 2012] and [Rodriguez-Martinez et al. 2011] in static power estimation.

The runtime computational overhead of our model mainly comes from computing Eq. (7) and is only 30 microseconds in the server we use. Our modeling framework is not intrusive in that it only takes estimated resource by existing query optimizer as inputs and can be implemented as a separate module in the DBMS.

7. CONCLUSIONS

This paper argues for the importance of building accurate and robust models for power cost estimation in database systems. For that purpose, we conducted system identification experiments on a single database server to explore the essential components of possible power models. Via such experiments, we revealed the fact that power consumption is correlated with the work size when the system utilization is low. Based on those findings, we proposed and evaluated a two-level power estimation model: we started from a series of physical models that describe the power cost of individual relational operators under a static system environment, and then used an online model estimation method to dynamically tune key parameters of the static models to achieve high robustness. The static models for important relational operators were built on empirical results obtained from running simple workloads using linear regression tools. The online model estimation scheme ensures that the model can tolerate system dynamics and fluctuations of workload characteristics. Performance of our models was validated by a large number of test cases that emulate realistic database runtime environment. In summary, our models are found to be effective – the estimation error is lower than 10% in almost all cases. We strongly believe our work has high technical significance in that it serves as the basis for power-aware query optimization – a key mechanism in

building energy-efficient database management system. Immediate follow-up work includes more experiments and the extension of the estimation model to all relevant relational operators. The idea is to apply similar system identification methods to those relational operators and integrate them into the composite model. Our estimation framework can also be extended in a few directions. For example, it is obviously meaningful to explore power modeling in a distributed database system, in which the power dynamics from storage and network systems are considerably large therefore cannot be ignored from the modeling process. Inspired by the success of our online model tuning method in power modeling, we are currently investigating the potential of such method in time estimation in traditional query optimizers. This can also be combined with our power models to explicitly quantify energy consumption of database systems.

REFERENCES

AGRAWAL, R., AILAMAKI, A., BERNSTEIN, P. A., BREWER, E. A., CAREY, M. J., CHAUDHURI, S., DOAN, A., FLORESCU, D., FRANKLIN, M. J., GARCIA-MOLINA, H., GEHRKE, J., GRUENWALD, L., HAAS, L. M., HALEVY, A. Y., HELLERSTEIN, J. M., IOANNIDIS, Y. E., KORTH, H. F., KOSSMANN, D., MADDEN, S., MAGOULAS, R., OOI, B. C., O'REILLY, T., RAMAKRISHNAN, R., SARAWAGI, S., STONEBRAKER, M., SZALAY, A. S., AND WEIKUM, G. 2009. The Claremont Report on Database Research. *Communications of ACM 52*, 56–65.

AHMAD, F. AND VIJAYKUMAR, T. N. 2010. Joint optimization of idle and cooling power in data centers while maintaining response time. *SIGARCH Comput. Archit. News 38*, 1, 243–256.

ALONSO, R. AND GANGULY, S. 1992. Energy Efficient Query Optimization. Tech. rep., Matsushita Info Tech Lab.

ASTRAHAN, M. M., BLASGEN, H. W., CHAMBERLIN, D. D., ESWARAN, K. P., GRAY, J. N., GRIFFITHS, P. P., KING, W. F., LORIE, R. A., MEHL, J. W., PUTZOLU, G. R., TRAIGER, I. L., WADE, B. W., AND WATSON, V. 1976. System r: Relational approach to database management. *ACM Transactions on Database Systems 1*, 97–137.

BENINI, L., BOGLIOLO, R., AND MICHELI, G. D. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems 8*, 299–316.

BERL, A., GELENBE, E., GIROLAMO, M. D., GIULIANI, G., DE MEER, H., DANG, M. Q., AND PENTIKOUSIS, K. 2010. Energy-efficient cloud computing. *Comput. J. 53*, 7, 1045–1051.

CHAUDHURI, S. 1998. An overview of query optimization in relational systems. In *PODS*. 34–43.

CHRISTODOULAKIS, S. 1984. Implications of certain assumptions in database performance evaluation. *ACM Trans. Database Syst. 9*, 2, 163–186.

GRAEFE, G. 2008. Database servers tailored to improve energy efficiency. In *Proceedings of the 2008 EDBT Wworkshop on Software Engineering for Tailor-made Data Management*. SETMDM '08. 24–28.

HARIZOPOULOS, S., SHAH, M. A., MEZA, J., AND RANGANATHAN, P. 2009. Energy efficiency: The new holy grail of data management systems research. In *CIDR*.

HAYAMIZU, Y., GODA, K., AND KITSUREGAWA, M. 2011. An Experimental Study on Energy Saving for Dynamic Voltage and Frequency Scaling in Online Transaction Processing. *Institute of Electronics, Information, and Communication Engineers (IEICE) Technical Report 110*, 162, 41 – 46.

IRANI, S., SINGH, G., SHUKLA, S. K., AND GUPTA, R. K. 2005. An overview of competitive and adversarial approaches to designing dynamic power management strategies. *IEEE Transaction on VLSI systems 13*, 12.

ISCI, C. AND MARTONOSI, M. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 36. IEEE Computer Society, Washington, DC, USA, 93–.

KANSAL, A., ZHAO, F., LIU, J., KOTHARI, N., AND BHATTACHARYA, A. A. 2010. Virtual machine power metering and provisioning. In *SoCC*. 39–50.

KARLIN, A. R., MANASSE, M. S., MCGEOCH, L. A., AND OWICKI, S. S. 1990. Competitive randomized algorithms for non-uniform problems. In *SODA*. 301–309.

KOOI, R. 1980. The optimization of queries in relational databases. Ph.D. thesis.

KUNJIR, M., BIRWA, P., AND HARITSA, J. 2012. Peak Power Plays in Database Engines. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*.

KURP, P. 2008. Green computing. *Commun. ACM 51*, 11–13.

LANG, W., KANDHAN, R., AND PATEL, J. M. 2011. Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. *IEEE Data Engineering Bulletin 34*, 1, 12–23.

LANG, W. AND PATEL, J. M. 2009. Towards eco-friendly database management systems. In *CIDR*.

LAWRIE, J. B. 2007. A brief historical perspective of the wiener–hopf technique.

MACKERT, L. F. AND LOHMAN, G. M. 1986. R* optimizer validation and performance evaluation for distributed queries. In *VLDB*. 149–159.

PALEOLOGO, G. A., BENINI, L., BOGLIOLO, A., AND MICHELI, G. D. 1998. Policy optimization for dynamic power management. In *Design Automation Conference*. ACM Press, 182–187.

POESS, M. AND NAMBIAR, R. O. 2008. Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results. *PVLDB 1*, 2, 1229–1240.

POESS, M. AND NAMBIAR, R. O. 2010. Tuning servers, storage and database for energy efficient data warehouses. In *ICDE*.

POESS, M. AND NAMBIAR, R. O. 2011. Power Based Performance and Capacity Estimation Models for Enterprise Information Systems. *IEEE Data Engineering Bulletin 34*, 1, 34–49.

POESS, M., NAMBIAR, R. O., VAID, K., STEPHENS, J. M., HUPPLER, K., AND HAINES, E. 2010. Energy benchmarks: a detailed analysis. In *e-Energy*. 131–140.

RIVOIRE, S., SHAH, M. A., RANGANATHAN, P., AND KOZYRAKIS, C. 2007. JouleSort: a balanced energy-efficiency benchmark. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 365–376.

RODRIGUEZ-MARTINEZ, M., VALDIVIA, H., SEGUEL, J., AND GREER, M. 2011. Estimating power/energy consumption in database servers. *Procedia Computer Science 6*, 0, 112 – 117.

SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G. 1979. Access path selection in a relational database management system. In *SIGMOD Conference*. 23–34.

TSIROGIANNIS, D., HARIZOPOULOS, S., AND SHAH, M. A. 2010. Analyzing the energy efficiency of a database server. In *Proc. of the international conf. on management of data*. SIGMOD '10. ACM, 231–242.

VAHIDI, A., STEFANOPOULOU, A., AND PENG, H. 2005. Recursive least squares with forgetting for online estimation of vehicle mass and road grade: theory and experiments. *Vehicle System Dynamics 43*, 1, 31–55(25).

WAAS, F. AND GALINDO-LEGARIA, C. 2000. Counting, enumerating, and sampling of execution plans in a cost-based query optimizer. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. SIGMOD '00. 499–509.

WANG, J., FENG, L., XUE, W., AND SONG, Z. 2011. A survey on energy-efficient data management. *SIGMOD Rec. 40*, 17–23.

WANG, Y., MA, K., AND WANG, X. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. *SIGARCH Comput. 37*, 314–324.

WANG, Y., WANG, X., CHEN, M., AND ZHU, X. 2008. Power-efficient response time guarantees for virtualized enterprise servers. In *IEEE Real-Time Systems Symposium*. 303–312.

XU, Z., TU, Y., AND WANG, X. 2010. Exploring power- performance tradeoffs in database systems. In *Proc. of ICDE*.

Appendix

I. DERIVATION OF STATIC MODEL PARAMETERS

Based on system identification conducted on the testbed introduced in Section 3, our system model is initialized with the following parameters via running experiments (results shown in Fig. 4):

$$\vec{W}(0) = \begin{bmatrix} 0.00206303 \\ 0.00271021 \\ 0.00238662 \end{bmatrix} \tag{8}$$

Those numbers are further refined in the following process. As discussed in Section 3.2, our model's purpose is to estimate the power cost with lowest error (i.e., the smallest EER). Also, the final static model mentioned in Section 4.3 is a mixed linear model for all the relational operators' power profiles. This model could be solved by the Linear Equation and System Solvers (LESS) in GAMS based on specially designed experiments for all operators mentioned in section 4.1. First, we conduct a series of experiments for each single relational operators. For example, for nested-loop join, we produce experiments on 100 simple two-table-join queries feeding into the PostgreSQL which is hinted to give nested-loop join execution plan. Based on the data collected from this experiment, we know the number of tuples fetched, the intermediate tuples for nested-loop join, and run-time power consumption for such operations. Such data is formatted to feed into LESS and it calculates the power coefficients for the assumable linear models that reach the maximum likelihood for the set of data obtained. Similar experiments are run for other relational operators. By that, we obtain a set of power coefficients for various relational operators. We use Weka[12] to find the best

[12] http://www.cs.waikato.ac.nz/ml/weka/

value for those power weight coefficients for the whole model. Finally, our static model is equipped with the following parameters:

$$\vec{W}(0) = \begin{bmatrix} 0.0023178 \\ 0.0024535 \\ 0.00223659 \end{bmatrix} \quad (9)$$

II. EER WORST CASE ANALYSIS

In analyzing the sources of errors and evaluating the accuracy of our models, it helps to answer the following question: *is there a provable upper bound of EER in the worst case?* The answer is yes. Using our experimental setup for an example, the maximum power consumption of our server (i.e., when all components are fully loaded) is about 190 watts and the idle power is about 110 watts when the CPU utilization is on the lowest level (Fig. 2). Let us consider two extreme cases: (i) our model predicts that the system runs at the maximal power while there is actually nothing running in the system at all. According to Eq. (5), EER in this case is (110-190)/110 = -72.7%. Clearly, the result is a serious overestimation; (ii) on the other hand, our model says the system is running on minimal power, but actually the system reaches its maximal capacity. The EER is now (190-110)/190 = 42.1%, and it reflects a dramatic underestimation. Thus, unlike query processing time estimation that could result in unbounded estimation error, the lower and upper bound of power cost prediction is bounded. Specifically, this bound is 72.7% under our hardware configuration.

III. ASSUMPTIONS

[Christodoulakis 1984] gives assumptions for time estimation models to lay out a common environment for research and practice of database query optimization. Similar to their work, we list some assumptions as the basis for building our static model.

(1) *Uniformity of per-tuple power cost*: the power usage for processing one tuple/indexed tuple in CPU is always the same.
(2) *Constant number of tuples per page*: the probability of referencing any page is $1/P$, where P is the number of pages.
(3) *Random replacement of tuples among pages*: the probability of referencing any tuple is $1/B$, where B is the number of tuples per page.

Assumption 1 affects whether there exist power coefficients in the power cost metrics. Assumptions 2 and 3 affect the amount of estimated resource (e.g., the cardinality of data table). By building models with the above assumptions, we will see decreased accuracy of estimation. However, such assumptions provide us with a starting point for power modeling without worrying about complexity of the system and workloads. When we build our online models based on the static model, the latter two assumptions are relaxed in order to enhance the robustness of our power models.